

126. (amended) The computer processor of claim 118, wherein a rule for altering the data storage content from the first calling convention to the second is determined by examining a descriptor associated with the instruction before the recognized execution flow or transfer.

REMARKS

This paper responds to the Office Action of October 1, 2002 (the "October Action"). The shortened statutory period runs through January 1, 2003. Accordingly, this response is timely.

Applicant respectfully requests reconsideration of the application. Claims 1-133 are now pending, a total of 133 claims. Claims 1-21 and 37-50 are allowed. Of the claims not allowed, claims 22, 51, 61, 87, 94, 96, 104 and 113 are independent. In many cases, the rejected claims are not explicitly discussed in the October Action; instead the written rejections are incorporated by reference from the Office Action of February 20, 2002 (the "February Action").

I. Entry of amendment after final rejection

As shown in the accompanying Request for Withdrawal of Finality, the October Action is prematurely final, and no showings under Rule 116 are required for entry of this Response.

II. Claims 104 and 113

Claim 104 is discussed in paragraphs 16 and 31(F) of the October Action. Claim 104 is unamended, and recites as follows:

104. A method, comprising the steps of:

executing instructions fetched from first and second regions of a single address space of the memory of a computer, the instructions of the first and second regions being coded for execution by computers of first and second architectures or following first and second data storage conventions, respectively, the memory regions having associated first and second modifiable indicator elements, a hardware structure for storing the indicator elements enforcing a requirement that the memory regions be necessarily disjoint, the modifiable indicator elements each having a value indicating the

architecture or data storage convention under which instructions from the associated region are to be executed;

when execution of the instruction data flows or transfers from the first region to the second, adapting the computer for execution in the second architecture or convention.

Paragraph 31(F) of the October Action admits that "Intel segments [as used in Richter '684] 'may' overlap." In contrast, claim 104 recites "enforcing a requirement that the memory regions be necessarily disjoint." "Necessarily disjoint," as recited in claim 104, necessarily excludes Richter's segments, which "may overlap." Claim 104 is therefor patentable over Richter '684.

Claim 113 recites an analogous limitation, and is patentable for analogous reasons.

Claims 105-112 and 114-133 are allowable therewith; further, these dependent claims recite further patentable limitations.

III. Overview of relevant portions of Richter '684

The October Action states that "there is no other way" that Richter could accomplish certain results, and that therefore Richter '684 must perform certain steps. *E.g.*, paragraph 31(D). As will be shown in this section III, the October Action makes three errors:

- There are other ways to accomplish the results that Richter '684 describes – and in many cases the "other ways" overlooked by the October Action lack the impractical features of the system design proposed in the October Action.
- Richter '684 does not solve the problem described in the October Action, nor does Richter '684 perform the steps described in October Action. Instead, Richter '684 teaches leaving the problem in place, and addresses a limited set of cases in which he can work around the problem.
- The analysis in the October Action is not a proper substitute for the obviousness analysis required by MPEP §§ 2143-2143.03 – to render a claim obvious, publication or patent prior art must actually teach how to practice the claim with a "reasonable expectation of success," not merely state that it could be done. *Paperless Accounting v. Bay Area Rapid Transit Sys.*, 804 F.2d 659, 665, 231 USPQ 649, 653 (Fed. Cir. 1986); MPEP § 2143.02. The October Action merely states that a solution to certain problems would be desirable, and therefore Richter '684 must have solved them. The MPEP forbids a rejection to be based on such reasoning. For example, in the case of anticipation, for something to be inherent, there must be no possible alternative. MPEP § 2112. A reference must be "enabling" to anticipate. MPEP § 2121. In the case of obviousness, a reference must teach a solution that offers a "reasonable expectation of success." MPEP § 2143.02. As will be shown in general in this section III, and below with respect to particular claims, the October Action fails to address the required issues.

As will be shown below, Richter's general approach is to modify the behavior of his RISC mode so that the incompatibilities with his CISC mode are reduced.

A. "Endianness" issues

The October Action reads a number of inferences into col. 9, lines 17-26 of Richter '684. This portion of Richter '684 addresses a "work around" for the conflict between "big endian" byte ordering and "little endian" byte ordering. Some types of computers (including the Intel X86, the CISC machine discussed in Richter '684) arrange data in memory in "little endian" fashion, so that the most-significant byte of any integer datum is at the higher address, and the least-significant byte is at the lower address. Other computers, including the conventional mode for PowerPC discussed in Richter '684, use "big endian" byte ordering, so that the most-significant byte of a datum is at the lower address. Thus, the following data structure, with the indicated hexadecimal values:

```
struct {
    4-byte integer a = x'11223344';
    2-byte integer b = x'5566';
    1-byte integer c = x'77';
    1-byte integer d = x'88';
    4-byte integer e = x'99aabbcc';
    8-byte char string = "character";
}
```

would be stored in memory in a little-endian machine, such as the Intel X86, as follows:

44	33	22	11	66	55	77	88	cc	bb	aa	99	c	h	a	r	a	c	t	e
----	----	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---	---	---	---

and this way in a big-endian machine:

11	22	33	44	55	66	77	88	99	aa	bb	cc	c	h	a	r	a	c	t	e
----	----	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---	---	---	---

Note that some of sections of this memory are reversed, and some must not be reversed if correct program execution is to be maintained.

The October Action asserts that "Richter et al. clearly teaches that the data storage content of the computer must be altered" from big-endian mode to little-endian. However, to do so would present insurmountable complexities and difficulties, and would be intolerably slow.

First, an implementation as proposed in the October Action would have to have some way to distinguish those memory cells that are to be reversed and those that must not be, and would have to identify the type of cell (4-byte-integer, 2-byte integer, character string, etc.) to

determine the size of each cell is that is to be reversed.¹ A program could not execute correctly if, for example, every four-byte chunk of the two pictures above were simply reversed.

Second, every time the system crossed from RISC to CISC, the system proposed in the October Action would have to visit every single cell of memory that was identified for reversal (in the physical memory, in the image paged out to the backing store, and in all files and databases accessed by the program), and reverse the memory contents cell-by-cell.

The inferences drawn in the Office Action are erroneous, and these consequences are unnecessary, in view of Richter's clear teaching of a contrary proposal. Col. 9, lines 10-26 describes Richter's scheme for avoiding the complex and time-consuming process proposed in the October Action. Richter defines certain code segments that use a "cross-breed" mode. In this mode, the computer decodes instructions using RISC instruction opcode definitions. However, Richter modifies his RISC memory load and store operations, to use the little-endian memory access convention of an X86 (col. 9, lines 22-26).²

- (a) Richter '684 nowhere states or suggests that any action is performed by the computer to convert one of the above data pictures to the other, and the October Action points to no language in Richter '684 to support an inference that such conversion occurs.
- (b) Richter '684 teaches no mechanism for distinguishing those blocks of memory that would have to be reversed and those that must not, and the October Action points to nothing in Richter '684 that even suggests such.
- (c) Richter '684 provides no mechanism for recognizing that the first four bytes of the above pictures are one block that is to be reversed, the next two bytes are another block that has to be reversed, the next two bytes are two different blocks that should not be reversed, the next four bytes should be reversed, and the last eight must not be reversed. The October Action points to nothing in Richter '684 in support of such an inference.

In a Memorandum of February 21, 2002 from Stephen G. Kunin, Deputy Commissioner for Patent Examination Policy to the Patent Examining Corps (the "Kunin memo"), Deputy Commissioner Kunin instructed as follows (emphasis added):

¹ The PowerPC may be operated in either big-endian mode or little-endian mode, but can only be switched between by a "hard reset." (See PowerPC RISC Microprocessor User's Manual, Exhibit 1, section 2.4.5. This strongly suggests that IBM and Motorola believed it to be essentially impossible to preserve correct program execution if two programs were to execute on a single processor with different endian modes. The suggestion that it would be obvious to switch a single program back and forth, mid-execution, is contrary to the knowledge in the art.

² The consequence, of course, is that in Richter's RISC-opcode-CISC-endian mode, the computer cannot correctly execute code produced by a conventional compiler for either machine.

It would not be appropriate for the examiner to take official notice of facts without citing a prior art reference where the facts asserted to be well known are not capable of instant and unquestionable demonstration as being well-known. ...

It is never appropriate to rely solely on "common knowledge" in the art without evidentiary support in the record, as the principal evidence upon which a rejection was based. As the court held in *Zurko*, an assessment of basic knowledge and common sense that is not based on any evidence in the record lacks substantial evidence support.

...

(3) If applicant challenges a factual assertion as not properly officially noticed or not properly based upon common knowledge, the examiner must support the finding with adequate evidence.

... If applicant adequately traverses the examiner's assertion of official notice, the examiner must provide documentary evidence in the next Office action if the rejection is to be maintained. If the examiner is relying on personal knowledge to support the finding of what is known in the art, the examiner must provide an affidavit or declaration setting forth specific factual statements and explanation to support the finding.

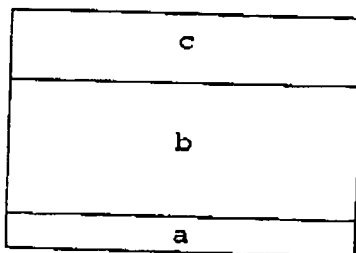
Applicant traverses all inferences drawn in the October Action that go above the explicit wording of col. 9, lines 17-26 of Richter '684. If any rejection relying on this portion of Richter '684 is maintained, Applicant requests (pursuant to 37 C.F.R. § 1.104(d)(2), MPEP § 2144.03, and the Kunin memo) a reference, or a citation to particular language in Richter '684, showing that each of the three mechanisms (a), (b) and (c) discussed in the previous paragraph was known in the art, and that there was motivation to modify Richter '684 in the manner proposed.

B. Parameter passing

Another incompatibility between two processing architectures may arise in the conflict between "calling conventions," for example, the ways parameters are passed in to a function and function values are returned. In the Intel X86, function parameters are pushed on the memory stack, in right-to-left order, with no padding. Thus, for a function with a C-language function with the following prototype:

```
double f( char a, long b, short c );
```

an Intel X86 might typically push two bytes of value c, then four bytes of b, then one byte of a, to create a parameter passing area on the stack that looks like this, with lower addresses at the bottom (in the Intel X36 architecture, stacks grow down from high addresses to lower addresses):



In contrast, in the PowerPC, parameters are stored in registers from left-to-right order, in registers r3-r10 and r13-r31.³ If a parameter is smaller than a register, the remainder of the register must be padded out to fill a register. Thus, the same parameters would be passed in a modern PowerPC (having 64-bit registers) as follows:

r3		r4		r5	
7 bytes of fill	a	4 bytes fill	b	6 bytes fill	c

Richter '684 nowhere states that he has any solution in mind for this mismatch between the two calling conventions, let alone teaching any mechanism for actually solving it:

- (d) Richter '684 never mentions any mechanism, applicable in a call from PowerPC to X86, for knowing how many parameters are in the registers, or whether to store 1, 2 or 4 bytes of any particular PowerPC register into the X86 memory, and neither Office Action even suggests that such a mechanism exists.
- (e) Richter '684 never mentions any mechanism, applicable in a call from X86 to PowerPC, for knowing how many bytes of parameter are on the stack, and neither Office Action even suggests that such a mechanism exists.
- (f) Assuming Richter's system knows that there are seven bytes of parameters on an X86 call stack (e.g., the example shown above), and the destination is coded in PowerPC, Richter '684 never mentions any mechanism for determining whether to move the seven bytes on the stack into one register or seven, and how to apportion the stack bytes into the registers. In considering questions (e) and (f), it should be recalled that conventional Intel hardware and conventional PowerPC hardware do not store any of this information – a calling convention is entirely enforced in software – and Richter '684 makes no mention of any unconventional hardware for this purpose.
- (g) The October Action points to nothing in Richter '684 that suggests that Richter '684 even considered any aspect of the cross-calling-convention problem, let alone developed an

³ A description of the PowerPC calling convention can be found at <http://www.ccs.neu.edu/home/will/com3355/Q13c/doc.cg.part1>.

operable solution. "Stacks" and "registers" are mentioned here and there throughout Richter '684, but never in the context of parameter passing, the central issue in any calling convention. The only mention of parameter passing is at col. 6, lines 17-18, where Richter '684 discusses the conventional mechanism for mismatched calling conventions, a software routine custom-coded for each individual case.

- (h) The October Action points to no data structure, hardware structure, or magic oracle that would provide any information to control the process of adjusting parameter storage in order to cross from one calling convention to the other. Richter's system leaves these problems entirely to the programmer.

If any rejection directed to any "calling convention" claim is maintained, Applicant requests (pursuant to 37 C.F.R. § 1.104(d)(2), MPEP § 2144.03, and the Kunin Memo) a reference, or a citation to particular language in Richter '684, discussing as many of items (d)-(h) as are relevant to any position taken in the rejection, a showing that a solution to the relevant problems were known in the art, and a showing of either "enablement" (for any § 102 rejection) or "reasonable expectation of success" (for any § 103(a) rejection).

C. Function return values

There is a third incompatibility between the X86 and PowerPC calling conventions. In the Intel X86, some function results are returned in the A register, and some in floating-point registers. In contrast, in PowerPC, all function results are returned in r3. In a return from an X86 routine to a PowerPC caller, Richter's system has no way to know whether to copy the A register into r3, or to copy a floating-point register – obviously it cannot copy both to the same destination. Richter '684 does not even describe this problem, let alone offer a solution.

If any rejection directed to "calling convention" claims is maintained, Applicant requests (pursuant to 37 C.F.R. § 1.104(d)(2), MPEP § 2144.03, and the Kunin Memo) a reference, or a citation to particular language in Richter '684, showing that a solution to the problem discussed in the previous paragraph was known in the art.

IV. Claims 22 and 96

Paragraphs 9 and 31(A) of the October Action purport to reject⁴ claim 22 under § 102(b). Claim 22 is unamended, and recites as follows:

⁴ Applicant notes that the October Action is incomplete with respect to claim 22, and thus no rejection properly exists. The October Action contains no element-by-element comparison of claim 22 to

22. A method, comprising the steps of:

executing instructions fetched from first and second regions of a memory of a computer, the instructions of the first and second regions being coded for execution by computers following first and second data storage conventions, the memory regions having associated first and second indicator elements, the indicator elements each having a value indicating the data storage convention under which instructions from the associated region are to be executed;

recognizing when program execution has flowed or transferred from a region whose indicator element indicates the first data storage convention to a region whose indicator element indicates the second data storage convention, and in response to the recognition, altering the data storage content of the computer to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention.

Paragraph 31(A) states that "Richter et al. clearly teaches that the data storage content of the computer must be altered." This reflects an erroneous reading of Richter '684. Instead, Richter '684 teaches a variety of techniques for avoiding the "execution flow" underlined in claim 22, and thus the "alteration of data storage content" is similarly avoided.

At col. 9, line 17-26, Richter '684 teaches a way to avoid "program execution [flowing] from a region [using] the first data storage convention to a region [using] the second data storage convention." This paragraph of Richter only discusses execution flow between two regions that observe the same storage convention. This paragraph only discusses crossing over from one little-endian segment to another little-endian segment. See section III.A, above. To the degree Richter addresses data storage conventions at all, he does so by avoiding execution flow from a region of one storage convention to another as recited in claim 22. Thus, there is no anticipation.

For this reason, claim 22 is patentable over Richter '684. Claims 54, 73, 76, 77 and 85 recite limitations that are analogous, and they are patentable for analogous reasons. Claims 23-36 are dependent on claim 22 and allowable therewith, and additionally for reasons recited in those individual claims.

the prior art, only an incorporation by reference of the February Action. However, as noted in Applicant's paper of June 2002, the February Action also failed to discuss the limitations of claim 22. The portion of the February Action purportedly directed to claim 22 stated only that claim 22 does "not teach or define above the invention claimed in claims 1-2." However, the "altering the data storage content" limitation discussed in Applicant's June paper does not appear in claim 1 or 2. Without a limitation-by-limitation comparison of claim 22 to Richter '684, neither Office Action raises a § 102(b) rejection of claim 22.

Claim 96 recites "program code in the first and second instruction sets using first and second different data storage conventions, respectively." As shown above, Richter '684 uses the same data storage convention for both instruction sets. Thus, claim 96 is not properly rejected over Richter '684. Claims 97-102 are allowable therewith, and are separately patentable for reasons recited in those claims.

V. Claims 51 and 61

Paragraphs 5 and 31(B) of the October Action discuss claims 51 and 61. Claim 51 is unamended, and recites as follows:

51. A method, comprising:

storing instructions in pages of a computer memory managed by a virtual memory manager, the instruction data of the pages being coded for execution by, respectively, computers of two different architectures and/or under two different execution conventions;

in association with pages of the memory, storing corresponding indicator elements indicating the architecture or convention in which the instructions of the pages are to be executed;

executing instructions from the pages in a common processor, the processor designed, responsive to the page indicator elements, to execute instructions in the architecture or under the convention indicated by the indicator element corresponding to the instruction's page.

In Applicant's June Response, it was pointed out that claim 51 recites storing indicator elements in association with "pages," and at best, Richter '684 discusses "segment descriptors."

Paragraph 31(B) of the October Action responds as follows:

[To] the extent defined by the exact meaning of the claim language, the reference to "pages" is merely an indication that there are sections or portions of memory managed by a memory manager.

The October Action fails to give proper consideration to the word "pages" recited in the claim.

To be "exact," claim 51 uses the word "pages," not the words "sections or portions." "Page" has been a well-established term of art for thirty years, and there are no synonyms in common use.⁵ MPEP § 2111.01 instructs that an examiner may not give novel definitions to well-established terms of art ("[T]he words of a claim must be given their plain meaning. In

⁵ Different manufacturers offer different extensions to the term "page," e.g., the TLB "granularity hint" of Digital Equipment Corp's Alpha processors, however, none known to Applicant overlap with Intel's definition of the word "segment."

other words, they must be read as they would be interpreted by those of ordinary skill in the art." emphasis added), particularly where there is no alternative available in the art. See MPEP § 2173.05(a) (where a claim is as clear as the language of the art permits, it is improper to require further clarification). Substitution of "sections or portions" for "pages" violates the "plain meaning" of the claim, and thus is not a "reasonable interpretation consistent with the specification" that can be used in framing a rejection. The rejection may be withdrawn.

Second, one of ordinary skill in the art would not consider the word "page" to be broad enough to cover Intel-type "segments." Richter himself contrasts "pages" and Intel-type "segments" at col. 6, lines 62-67 (noting that Intel-type segments need not be page aligned).

Third, the October Action misparaphrases claim 22 by omitting the word "virtual" from the phrase "memory manager." Intel segments are not part of the virtual memory management system; rather they are part of the memory protection system. (For example, segments were part of the Intel 80286, but virtual memory paging was not added until the 80386.)

Thus, the "pages of a computer memory managed by a virtual memory manager" cannot be met by the Intel "segments" of Richter '684. If the Examiner is aware of a reference that uses the terms "pages ... managed by a virtual memory manager" to cover Intel-type segments⁶, and can establish that this usage is more common in the context of Intel and Intel-clone designs than the distinction drawn in Richter '684, a copy of such reference is requested, and Applicant will respond as appropriate. However, in absence of substantial evidence that the term "pages of a computer memory managed by a virtual memory manager" has in fact been used in the art as a synonym for Intel-type segments, MPEP § 2111.01 requires that the term be given its ordinary meaning in the art.

Because claim 51 recites an element that is absent from Richter '684, claim 51 is patentable over that reference. Claims 30, 61, and 89 recite analogous language and are patentable for analogous reasons. Claims 31, 32, 52-60, 62-86, 90 and 91 are dependent on these claims, and allowable therewith. These claims also recite further patentable features.

⁶ Paragraph 31(F) of the October Action suggests that the Examiner is aware of the different definition of "segment" as used in connection with Intel products (and Richter's implementation of the Intel architecture), and the entirely different definition of "segment" used in most of the rest of the computer literature. Because Richter '684 clearly indicates that the word "segment" is used in the Intel sense, it would be inappropriate to present a reference that used the word "segment" in the non-Intel sense in response to this request.

VI. Claim 87

Paragraphs 6, 16 and 17 of the October Action assert that claim 87 is obvious over Richter '684. Claim 87 is unamended, and reads as follows:

87. A method, comprising the steps of:

executing a control-transfer instruction under a first execution context of a computer, the instruction being architecturally defined to transfer control directly to a destination instruction for execution in a second execution context of the computer;

before executing the destination instruction, altering the data storage content of the computer to establish a program context under the second execution context that is logically equivalent to the context of the computer as interpreted under the first execution context, the reconfiguring including at least one data movement operation not included in the architectural definition of the control-transfer instruction.

The claim recites a genus, a set of methods in which a computer "[alters] the data storage content of the computer to establish a [second] program context ... that is logically equivalent to the ... first execution context, the reconfiguring including at least one data movement operation not included in the architectural definition of the control-transfer instruction." If this claim is to be rejected under § 103(a), then MPEP § 2143.02 requires that an Office Action show that the prior art teaches at least one species within the genus with a "reasonable expectation of success." There is no need for the claim to recite any particular steps for carrying out the recited process, or any other characteristic of any particular species within the genus, if no species of the genus is taught by Richter '684.

The October Action makes no attempt to show this "reasonable expectation of success." That alone is sufficient to render the rejection incomplete.

The October Action's discussion of claim 87 (paragraphs 31(C) and (D)) contains no citation to any particular teaching of Richter '684; rather, the October Action relies entirely on conjecture as to what Richter '684 might have done. Conjecture is not "substantial evidence." Rejections that lack "substantial evidence" are forbidden. See Kunin Memo. In any future rejection, Applicant requests a particular designation of particular language in prior art references in support of each limitation of any rejected claim. In particular, Applicant requests "substantial evidence" showing the underlined portions of claim 87.

Finally, paragraph 31(D) states that "there is no other way to convert from RISC data structures to CISC data structures without movement of data in some manner." The flaw in this

reasoning is that Richter '684 does not convert "from RISC data structures to CISC data structures." Instead, as discussed above in sections III.A (Richter's approach to "endianness") and IV (claim 22), Richter '684 alters his RISC machine to use CISC data structures.

Claims 88-93 are dependent on claim 87, and allowable therewith. These claims also recite further patentable features.

VII. Claim 94

Paragraphs 5 and 31(E) of the October Action discuss claim 94 in light of Richter '684, col. 6, lines 27-39; col. 9, lines 26-57; and col. 14, lines 33-43. As now amended, claim 94 recites as follows:

94. A method, comprising the steps of:

executing a section of computer object code twice, without modification of the code section between the two executions, the code section materializing a destination address into a register and being architecturally defined to directly transfer control indirectly through the register to the destination address, the two executions materializing two different destination addresses;

the two destination code sections at the two materialized destination addresses being coded in two distinct instruction sets and, respectively, obeying the default calling conventions native to each of the two instruction sets, neither instruction set being a subset of the others.

As shown in section 2.4.5 of the PowerPC RISC Microprocessor User's Manual (Exhibit 1), "the default mapping for PowerPC processors is big-endian" The only mode available in Intel X86 processors is little-endian. The portions of Richter '684 indicated in the October Action discuss only changing instruction set. None of these portions even mention the problems raised when each of the two instruction sets uses its own default native calling convention.

Thus, the § 102 rejection may be withdrawn. Further, from the little Richter '684 says about other issues, it appears that Richter's approach to function calls is likely analogous to the mechanism he uses to deal with "endianness" issues (section III.A of this paper): Richter apparently imposes a CISC calling convention on his RISC code, creating a non-default, non-native data storage convention for his RISC (col. 9, lines 18-26).

Paragraph 26 of the October Action asserts that certain technologies are "traditional." "Traditional" is not mentioned in the MPEP as an alternative test for either inherency or

anticipation, especially in view of the existence of alternatives.⁷ If any rejection is maintained, Applicant requests "substantial evidence" as required by the Kunin Memo, applied as required by the MPEP.

VIII. Dependent claims

Many dependent claims are rejected over the art. However, section III of the accompanying Request for Withdrawal of Finality gives several examples of cases in which the October Action fails to compare limitations recited in many of these claims to the prior art. Such piecemeal examination is discouraged by 37 C.F.R. § 1.105 and MPEP § 707.07(g). It is requested that any future Office Action indicate the allowability of any claim that recites a limitation against which no prior art is cited.

In view of the amendments and remarks, Applicant respectfully submits that the claims are in condition for allowance. Applicant requests that the application be passed to issue in due course. The Examiner is urged to telephone Applicant's undersigned counsel at the number noted below if it will advance the prosecution of this application, or with any suggestion to resolve any condition that would impede allowance. In the event that any extension of time is required, Applicant petitions for that extension of time required to make this response timely.

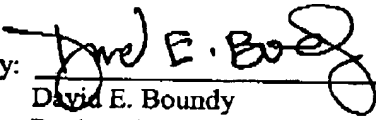
⁷ In addition to Richter's express disclosure of CISC data storage conventions for RISC instructions, the Examiner's attention is further drawn to an alternative embodiment disclosed in Applicant's specification at section IV, pages 66-67. There, Applicant discloses that some RISC code may use CISC calling conventions for parameter passing and function return values. This technique would be consistent with the other techniques disclosed in Richter '684, and Richter '684 says nothing to the contrary.

Kindly charge any additional fee, or credit any surplus, to Deposit Account 50-0675,
Order No. 5231.03-4000.

Respectfully submitted,

SCHULTE ROTH & ZABEL, LLP

Dated: December 2, 2002

By: 
David E. Boundy
Registration No. 36,461

Mailing Address:
SCHULTE ROTH & ZABEL, LLP
919 Third Avenue
New York, New York 10022
(212) 756-2000
(212) 593-5955 Telecopier

Exhibit 1 Excerpts from PowerPC 601 RISC Microprocessor User's Manual

REWRITTEN CLAIMS MARKED UP TO SHOW CHANGES

1 1. (twice amended) A computer, comprising:
2 a processor pipeline designed to alternately execute instructions coded for first and
3 second different computer architectures or coded to implement first and second different
4 processing conventions;
5 a memory for storing instructions for execution by the processor pipeline, the
6 memory being divided into pages for management by a virtual memory manager, a single
7 address space of the memory having first and second pages;
8 a memory unit designed to fetch instructions from the memory for execution by the
9 pipeline, and to fetch stored indicator elements associated with respective memory pages of
10 the single address space from which the instructions are to be fetched, each indicator element
11 designed to store an indication of which of two different computer architectures and/or
12 execution conventions under which instruction data of the associated page are to be executed
13 by the processor pipeline, the indicator elements being architecturally addressable when the
14 processor pipeline is executing under one of the architectures or data storage conventions,
15 and architecturally unaddressable when the processor pipeline [computer] is executing under
16 the other architecture or data storage convention;
17 the memory unit and/or processor pipeline further designed to recognize an execution
18 flow from the first page, whose associated indicator element indicates the first architecture or
19 execution convention, to the second page, whose associated indicator element indicates the
20 first architecture or execution convention, and in response to the recognizing, to adapt a
21 processing mode of the processor pipeline or a storage content of the memory to effect
22 execution of instructions in the architecture and/or under the convention indicated by the
23 indicator element corresponding to the instruction's page.

27. (amended) The method of claim 22, wherein
a rule for copying data from the first location to the second is determined by
examining a descriptor associated with the instruction [location of execution] before the
recognized execution flow or transfer.

59. (amended) The method of claim 54, wherein
a rule for copying data from the first location to the second is determined by
examining a descriptor associated with the instruction [location of execution] before the
recognized execution flow or transfer.

93. (amended) The method of claim 87, wherein
a rule for copying data from the source memory or register to the destination register
or memory is determined by examining a descriptor associated with the address [location] of
the control-transfer instruction.

1 94. (amended) A method, comprising the steps of:
2 executing a section of computer object code twice, without modification of the code
3 section between the two executions, the code section materializing a destination address into
4 a register and being architecturally defined to directly transfer control indirectly through the
5 register to the destination address, the two executions materializing two different destination
6 addresses;
7 the two destination code sections at the two materialized destination addresses being
8 coded in two distinct instruction sets and, respectively, obeying the default data storage
9 conventions native to each of the two instruction sets, neither instruction set being a subset of
10 the others.

Application Serial No. 09/385,394
Atty. Docket No. 5231.03-4000

97. (amended) The computer processor of claim 96, wherein the memory unit and software arc designed to effect a transition between instruction boundaries, between execution in a region coded in the first instruction set using the first data storage [calling] convention to execution in a region coded in the second instruction set using the second data storage [calling] convention, so that code at the source of the flow or transfer may effect the execution transition without being specially coded for code at the destination.

98. (amended) The microprocessor chip of claim 96, wherein the two data storage conventions are first and second [two] calling conventions.

100. (amended) The microprocessor chip of claim 98 [96], further comprising:
software and/or hardware designed to copy a datum from a first location to a second location, the first location having a use under the first calling conventions analogous to the use of the second location under the second calling conventions.

102. (amended) The computer processor of claim 98 [96], wherein
a rule for altering the data storage content from the first calling convention to the second calling convention is determined based on an instruction at the location of execution at the source of the recognized execution flow or transfer.

103. (amended) The computer processor of claim 98 [96], wherein
a rule for altering the data storage content from the first calling convention to the second calling convention is determined by examining a descriptor associated with the instruction [location of execution] before the recognized execution flow or transfer.

126. (amended) The computer processor of claim 118, wherein
a rule for altering the data storage content from the first calling convention to the second is determined by examining a descriptor associated with the instruction [location of execution] before the recognized execution flow or transfer.